LA-UR-01-4087

Approved for public release;
distribution is unlimited.

Title: An Approach to Extreme-Scale Simulation of Novel Architectures

Author(s): Francis J. Alexander
Kathryn Berkbigler
Graham Booker
Brian Bush
Kei Davis
Adolfy Hoisie

Submitted to: LACSI 2001 Symposium
El Dorado Hotel
Sante Fe, NM
15-18 October 2001

An Approach to Extreme-Scale Simulation of Novel Architectures*

Francis J. Alexander, Kathryn Berkbigler, Graham Booker,
Brian Bush, Kei Davis, and Adolfy Hoisie
Los Alamos National Laboratory
Los Alamos, NM 87545

Abstract

We outline an approach for simulating computing architectures applicable to extreme-scale systems (thousands of processors) and to advanced, novel architectural configurations. We believe that simulation is the predictive tool of choice for evaluating the performance of such systems. Our component-based design allows for the seamless assembly of architectures from representations of workload, processor, network interface, switches, etc., with disparate resolutions into an integrated simulation model. This accommodates different case studies that may require different levels of fidelity in various parts of a system. Our initial prototype, comprising low-fidelity models of workload and network, aims to model at least 4096 computational nodes in a fat-tree network. It supports studies of simulation performance and scaling rather than the properties of the simulated system themselves. Future work will allow more realistic simulation and visualization of ASCI-like workloads on very large machines.

1 Motivation

The magnitude of the scientific computations targeted by the ASCI project requires as-yet unavailable computational power. To facilitate these computations ASCI plans to deploy massive computing platforms, possibly consisting of tens of thousands of processors, capable of achieving 10-100 tera-ops. For various reasons the current approach to building a yet-larger supercomputer—connecting commercially available SMPs with a network—may be reaching practical limits. In response the DOE Advanced Architecture Initiative seeks to research alternative high-performance computing architectures.

The path to better hardware design and lower development costs involves performance evaluation, analysis, and modeling of parallel applications and architectures, and in particular *predictive* capability. Performance studies are routinely used to select the best architecture or platform for a given application, select the best algorithm for solving a particular problem, and to study scalability with respect to problem and platform size. Evaluating and analyzing the performance is challenging, primarily because of the large number of components making up such systems and the complex interactions that occur between them.

The tools of the trade in performance modeling and analysis are typically categorized as algorithmic/analytical analysis, statistical analysis, analysis with queuing theory, and simulation. Depending on the problem one or more or these methods will be more appropriate than others. Although significant results have been obtained in recent work for an important class of applications of interest to ASCI [1, 2], analytical modeling of systems and applications of this scale is not always

*This work was carried out under the auspices of the Department of Energy at Los Alamos National Laboratory under ASCI DisCom and LDRD-ER 2000047.

possible. Queuing models generally lead to very complex nonlinear equations whose solution is intractable. For systems of ASCI-proposed size and complexity *simulation* remains the predictive tool of choice, though simulation may be augmented by analytical and statistical analysis.

Three related targets for our simulation effort have been identified: simulation of ASCI-scale parallel systems using a realistic ASCI workload, simulation of ASCI scale storage systems and I/O, and simulation of the high-performance ASCI wide-area network. All of these aspects of ASCI system design are equally important and tractable by the approach we propose. However, given the scale of the effort required, we envision a staged approach to tackling these problems.

In conjunction with the other methodologies, the proposed simulation environment could be used for

- exploration of hardware/architecture design space;
- exploration of algorithm/implementation space both at the application level (e.g. data distribution and communication) and the system level (e.g. scheduling, routing, and load balancing);
- determining how application performance will scale with the number of processors or other components;
- analysis of the tradeoffs between performance and cost;
- testing and validating analytical models of computation and communication such as **LogGP** [3] and **BSP** [4].

A canon of the field of performance evaluation is that hardware and software performance are inextricable—hardware performance is meaningful only in the context of applications—thus these capabilities are not entirely independent.

2 Goals

The general goal is to design and implement a simulation framework for design and analysis of extreme-scale parallel and distributed computing systems, and as an ongoing part of this process to validate the accuracy of results characterized by any particular model. An intermediate goal is to model (and validate the model of) the ASCI Q machine [5] with a realistic ASCI workload.

We take as given that it is not feasible to simulate an extreme scale machine and workload with perfect fidelity; on the other hand in certain circumstances it may be desirable to simulate some subset of such a machine with near-perfect fidelity. In any case the simulator itself should be able to exploit a machine of arbitrary size. Once defined, representations of logical *components* (e.g. processors or SMPs, network switches and interconnects, programs or workloads, etc.) should be easily assembled into differing configurations. Portability is essential. In more detail, the simulation system should

- be scalable to model systems comprising 10,000 processors or more;
- allow arbitrary sets of components to be represented with arbitrary degrees of fidelity, in terms of both structure (e.g. comprising distinct subcomponents) and timing;
- allow arbitrary (meaningful) configuration of components;

- allow description of the machine configuration to be as independent as possible from of the descriptions of the components;
- be genuinely portable across platforms ranging from single-processor workstations to clusters of SMPs;
- be able to interface to other distinct applications such as direct execution simulators and visualization systems.

These requirements suggest factoring the simulator into three parts: component descriptions, configuration descriptions, and an underlying, reasonably generic, and reasonably light-weight simulation system with which all porting issues are associated. An object-oriented approach facilitates these goals.

It is clear that simulating systems of the size and complexity that we envision will require the use of parallel simulation [6]. Furthermore, the parallel simulation substrate must support composition of simulations and be very efficient in its implementation. We concluded that a conservative synchronization scheme would have the best chance of success for this application. The requirement of portability across a variety of platforms led us to a parallel simulation substrate that runs on both shared memory and distributed memory machines. The Scalable Simulation Framework (SSF) [7] and the implementation of this framework being developed at Dartmouth College, DaSSF [8, 9], is our current choice.

3 Initial Prototype

To determine the suitability of DaSSF as a parallel simulation substrate a small prototype model was built. The purpose was to allow us to become familiar with DaSSF and to gain experience in constructing models. The prototype was to be a learning experience and feasibility study rather than the basis for conducting a specific simulation study.

We used the results of the domain analysis to define a subset of components to implement in the prototype. We chose not to model the processors and memory hierarchy of an SMP node in any detail and instead to focus on modeling the interconnection network between nodes as a fat-tree network with a circuit-switched routing protocol. For the workload, rather than model the message traffic from any specific application, we chose to have each processor node emit messages with exponentially distributed interarrival times. The message destination node is selected with uniform probability, and the message size is exponentially distributed. The parameters for the distributions are inputs to the simulation.

3.1 Requirements

Our requirements for the prototype were that it exercise all the essential components of DaSSF and several of the DaSSF extensions to SSF. To investigate the scalability of DaSSF itself the components of the prototype were to be such that they could could be easily configured into arbitrarily large models. Since we were not conducting a real performance study, we were not concerned with modeling our system components with high fidelity, but rather determining whether DaSSF was an appropriate substrate for developing components with arbitrary levels of fidelity. A small model that used all the features was desirable for rapid development. At the same time we were also interested in the scaling properties of DaSSF since we will want to develop very large models in the future. The data collection capabilities provided by DaSSF were another topic to be investigated.

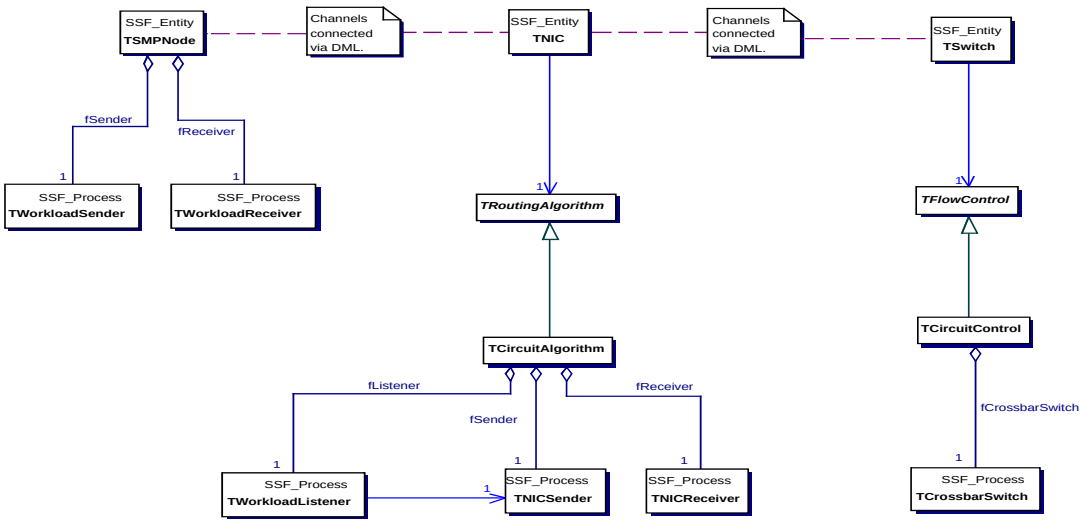

Figure 1: Simplified UML (Unified Modeling Language) class diagram for the simulation prototype.

DaSSF is provided for several platforms and may be compiled with the native compilers or with g++. We tested our model on a variety of platforms using native compilers and vendor-optimized versions of MPI. We were also interested in the capability of integrating DaSSF with our own C++ classes as well as with standard components such as the Standard Template Library. The ease of debugging simulations that use DaSSF was also to be evaluated.

3.2 Design

The DaSSF API provides five base classes that applications may subclass. The `SSF_Entity` class defines the entities in the simulation. The `SSF_Process` class defines the behaviors that entities possess. Entities are connected to each other via channels, an `SSF_OutChannel` in the transmitting entity and an `SSF_InChannel` in the receiving entity. An `SSF_Event` represents the information that flows between entities across the channels. Parameterized properties of model components, the number to be instantiated, and their connectivity are specified via an input file written in the Domain Modeling Language (DML).

A simplified UML (Unified Modeling Language) diagram of our prototype model is shown in Figure 1. The model contains three types of entities, representing the SMP node, the network interface card (NIC), and the network switch.

The SMP node has an outgoing channel for sending messages to its NIC and an incoming channel for receiving messages from its NIC. The NIC has a corresponding incoming channel for receiving messages from its SMP node and an outgoing channel for sending messages to its SMP node. Additionally, the NIC has an outgoing channel and an incoming channel that connect it to its network switch. Each network switch has 8 incoming channels and 8 outgoing channels. At the level nearest the NICs, 4 of the channels communicate with 4 NICs and 4 communicate with the

next level in the fat tree. (An example of a fat-tree network is shown in Figure 2). Higher in the fat tree communication involves only other switches. The route that a message takes through the network is determined at the source and is the same for all packets in a message. The route may be read and stored in a routing table or computed via a routing method.

The SMP node has two processes, `TWorkloadSender` for sending messages and `TWorkloadReceiver` for receiving messages. The NIC has a routing algorithm which has three processes, `TWorkloadListener` for receiving messages from the source SMP and buffering them if the NIC is busy, `TNICSender` for splitting the message into packets and sending the packets to the network switch, and `TNICReceiver` for receiving packets from the switch, reassembling the message and sending it to the destination SMP. The network switch has a flow control algorithm which has one process, `TCrossbarSwitch`, that receives packets on an incoming channel from a NIC or a another switch and forwards them out the appropriate outgoing channel to the next switch or NIC as specified by the route that is embedded in the packet.

We defined separate processes for the actions performed by the SMP node and NIC entities as a way of modularizing the logic, to reflect the fact that a NIC may represent a significant process running asynchronously with the SMP, and to allow for the possibility of multiple NICs per SMP. However this necessitates the use of other mechanisms such as a semaphore or an internal channel to allow processes belonging the same entity to communicate with each other. Rather than attaching the processes for message and packet handling directly to the NIC, we interposed the abstract `TRoutingAlgorithm` object and created a `TCircuitAlgorithm` class that contains the processes which implement a circuit routing algorithm for packet delivery. This gives us the flexibility to define other algorithms in the future and give new behavior to the NIC by simply selecting at runtime a different algorithm to be constructed. Similar logic pervades the design of the switch and its process.

Two types of events are defined, one for a message, and one for the packets in a message. Packets are further subclassed into four types: the open circuit packet, the acknowledge circuit packet, the close circuit packet, and the data packet. The precise behavior that occurs in the processes depends upon the type of packet that arrives.

Our prototype employs the DaSSF-supplied random number generation module. We collect traces of packet movement through the network using DaSSF's data packing and dumping capability.

3.3 Implementation

The present implementation of our prototype simulator consists of approximately five thousand lines of ANSI- and POSIX-compliant C++ that runs using MPI under the Linux, Solaris, and Irix operating systems. We use DaSSF's built-in user threads to avoid the operating-system overhead associated with creating hundreds of native threads. We have executed the code on single processor boxes, SMPs, and clusters of workstations on a LAN. Since the specifications of modelled components and architectures are entirely independent of the host platform models may be developed and executed on any convenient or appropriate host.

The data-collection facility in the implementation permits one to collect detailed information concerning the history of each simulated message: its movement from the computational node to the network interface card, the opening of a network circuit for its transmission, its packetization, its acknowledgement, and the closing of its network circuit. The implementation has been tested to verify correctness of time delays, probabilistic distributions, and synchronization behaviour. The data-collection will also support our visualization capability, allowing users to view message traffic in the simulated system.

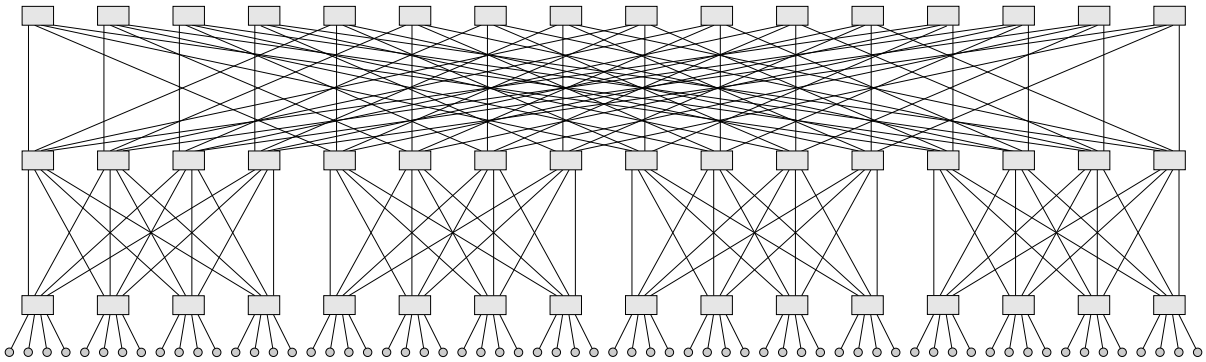

Figure 2: A simulation model with 64 computational nodes and 48 switches. Each rectangle represents a 8-port switch and each circle represents a computational node and its network interface card.

3.4 Results

We have constructed a variety of architecture models for testing the simulation and for measuring its performance and scalability. The simplest models consist of computational nodes connected to each other by a bus or via a pair of network interface cards; slightly more complex models contain one or two network switches with eight computational nodes. These small models provide a testbed for verifying that the timing delays in various parts of the simulation match those of the hardware being simulated. They also supply a convenient platform for debugging and detailed tracing of the simulation's progress. Even a simulation of a small system executing for a short time can produce a large amount of data if the history of each simulated entity, process, and event is recorded.

Our larger models contain 64, 128, ..., up to 4096 computational nodes networked together in a fat-tree topology. Figure 2 shows the layout of a 64-node system. A 4096-node system (similar in size to a proposed ASCI Q machine architecture) uses these 64-node systems as building blocks—the nodes of one such system are each replaced with a 64-node system. Having these more realistically-sized models allows us to undertake meaningful studies of system performance and scaling—both of the simulation system itself and of the architectures being simulated. Our current studies focus on the behavior of the simulation system rather than the simulated architecture.

Figure 3 shows the results of a simulation of the 64-node, 48-switch architecture shown in Figure 2, with a load of 100 MB/s of message traffic originating at each node. The diagram shows that many of the messages reach their destination in the minimum possible time (based on network connectivity and time delays), but that a significant fraction of the messages have additional delays due to network congestion. The mean message size is 4 KB. The simulation is not realistic in the sense that the network protocols do not correspond to those of any actual network hardware and the workload is highly generalized. Nevertheless, the simulation does show typical characteristics for a moderately-loaded system.

Based on our early simulation experiments, we expect to be able to simulate 4096-node clusters using computing platforms such as the SGI Origin 2000; we predict peak memory usage to be 8 GB in our initial prototype. Table 1 shows some of the performance figures we have obtained so far for a 64-node cluster. Note that performance degrades as the number of computational nodes increases because the processing nodes are not performing computational work; essentially only message passing is taking place. Future simulations involving the direct execution of actual applications will have a high proportion of their CPU time spent on the workload representation,

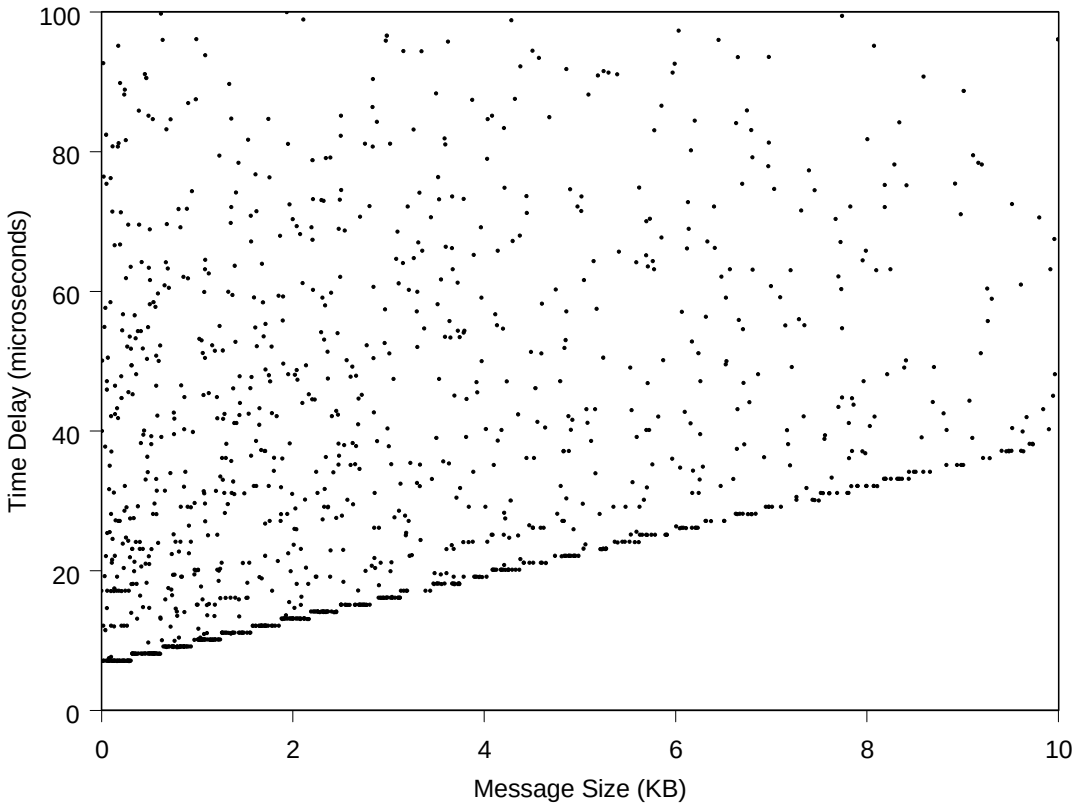

Figure 3: Example output data from a 64-node simulation where each node generates an average of 100 MB/s of message traffic on the network. The mean message size is 4 KB.

Platform	Computational Nodes	Events (1/sec)	Execution Time (sec)
Linux, 733 MHz Pentium III	1	2020.2	178.1
	2	689.9	521.6
Linux, 500 MHz Pentium III	4	494.3	728.0
	8	379.9	947.3
Solaris, Sun SPARC Ultra 5	1	1105.1	325.6
Irix, Origin 2000	1	1298.3	277.2
	4	1351.1	266.4
	16	630.2	571.1

Table 1: Performance of the initial prototype simulating at 64-node architecture on a variety of computing platforms.

and hence will exhibit more favorable scaling characteristics.

4 Future Directions

Our simulation effort will proceed along several tracks. Our primary future undertaking is the enhancement of the basic architecture simulation framework. In parallel, we are developing capabilities for the direct execution of application workload within our simulation and the visualization of the simulation and its results. Finally, we plan to augment our simulation with analytic and statistical analyses of workload and of simulation output using both traditional and novel approaches.

4.1 Simulation Framework

Our iterative, component-based approach to developing an architecture simulation allows us flexibility in choosing where to apply future effort. The initial prototype models an architecture at a coarse level. Based on our analysis of this simulation and consultation with domain experts we can determine the parts of the simulation where fidelity most profitably can be enhanced. Initial indications are that we need a relatively low fidelity representation of network switches and protocols, but that higher fidelity is necessary for the workload and processor cache representations. We plan to focus on developing a direct execution capability in the near future.

Our component-based development process will enable us to seamlessly compose hardware, protocols, and workloads of varying fidelities into a single simulation. We have begun investigating the possible use of hardware description languages to formally specify architectural configurations. This will facilitate the quick assembly of architectures for case studies.

We are completing the evaluation of DaSSF for use as the underlying discrete-event-handling substrate of the simulation framework. Depending upon the results of this study we may decide to evaluate additional discrete-event systems. Our high-level software design will make it relatively painless to substitute a different discrete-event system for DaSSF if necessary. Identifying high-performance and scalable discrete-event handling mechanisms is critical for the project's success.

4.2 Direct Execution

The initial prototype uses randomly sized messages at random intervals to simulate the workload on the SMP node. This was adequate for the purposes of the prototype but we recognize the need for a much more accurate representation of workload for most simulations. We intend to experiment with direct execution as a means of simulating real ASCI workloads.

In direct execution the application is executed on the same machine used to perform the simulation. The application is typically modified to call the simulator only for those operations that differ between the host machine and the simulated machine. Using the host machine to directly execute some instructions rather than simulating all instructions can result in considerably faster execution with minimal loss of accuracy when the host and target have similar architectures.

Most ASCI applications of interest use MPI (Message-Passing Interface) for communication between parallel processes. Our initial prototype could be augmented to directly execute the computations occurring on a node and invoke the simulator when calls to the MPI library occur. Timings for the directly executed statements can be obtained and added to the simulated time maintained by the simulator as it tracks the passage of the MPI message through the interconnection network. The coupling of direct execution with the DaSSF simulation substrate will be investigated in the near term.

A second approach to direct simulation is to run the program once and obtain accurate timing information between the blocks of interest. This method allows for faster simulation of the machine because the simulation only needs to know the message size and time sent. The greatest disadvantage of this method is that it does not account for cases where the time of message receipt affects the behavior of the program [12].

4.3 Visualization

We are also pursuing visualization of these simulations. We will focus on both visualizing the execution of the simulation and on visualizing the performance of the simulated system. Visualization will also aid in debugging the simulation itself, in developing and evaluating the efficiency of load balancing of the simulation entities, and in understanding synchronization between simulation timelines. Visualizing the simulated system will allow users to understand how varying workload or network architecture affects the overall performance of an advanced or novel architecture.

4.4 Statistical Characterization

The theoretical and computational issues involved in large-scale network modeling have direct analogues in some of the most challenging and important problems in statistical physics. The complicating features which appear in both contexts include multiple spatial and temporal scales, and strongly correlated dynamics and stochasticity. As a result we shall bring to bear on problems arising in network performance modeling techniques originally developed and already highly successful in the context of nonequilibrium statistical physics.

One such technique is a numerical Rayleigh-Ritz method [13]. This is a variational formulation for the time dependence of the probability distribution functions of network variables. Whether one uses a discrete event dynamical system or stochastic fluid level approach, the network variables are described by a large system of nonlinear, coupled, stochastic equations. Due to the nonlinear nature, these equations cannot be solved exactly and require that approximations be made. The approximations which describe higher order statistics in terms of lower ones are known as a closure. The Rayleigh-Ritz variational method then tests these statistical closure ideas using the exact

dynamics (i.e., the original equations), but more cheaply and quite often under more extreme circumstances than is feasible by direct numerical simulation.

Closure schemes can then be developed by using educated guesses for the system statistics. These 'guesses' may be inspired from an analysis of the data using the visualization tool. As a result, empirical data from actual networks and workloads may be exploited in a numerical calculation. Additional statistical information can be calculated within the variational formulation, including multi-time correlations, and the probability of large fluctuations in performance. Moreover, there are internal consistency checks available which may be used as diagnostics to detect *a priori* faulty predictions of the closures, potentially reducing the amount of time spent on inadequate models.

Using this methodology we will attempt to address several types of problems such as the probability of buffer overflow and likelihood of long-time performance averages. Moreover, we expect that well chosen closures will lead to being able to answer these questions with greatly reduced computational effort.

5 Acknowledgements

Thanks to members of the extended LANL ParSim team for technical input, including Mike Boorman, Fabrizio Petrini, Steve Smith, and Harvey Wasserman; and to David Nicol and Jason Liu (Dartmouth College), Josep Torrellas (University of Illinois at Urbana-Champaign), and Tom Caudell (University of New Mexico). Anand Sivasubramaniam (Pennsylvania State University) and Madhav Marathe participated in the initial project proposal.

References

[1] "Scalability Analysis of Multidimensional Wavefront Algorithms on Large-Scale SMP Clusters," *Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computations, Frontiers* February (1999).

[2] A. Hoisie, O. Lubeck, H. Wasserman, "Performance Analysis of Wavefront Algorithms on Very-Large Scale Distributed Systems," Lecture Notes in Control and Information Sciences 249, Springer-Verlag, (1999).

[3] D.E. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eiken, *LogP: Towards a Realistic Model of Parallel Computation.* Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Diego, CA, May 1993.

[4] M. Goudreau, K. Lang, S. Rao, T. Suel, and T. Tsantilas, *Towards Efficiency and Portability: Programming with the BSP Model* Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures, Padova, Italy, 1996.

[5] R. Kaufman, *The Q Supercomputer and Compaq.* High Performance Technical Computing News, Issue 18, November 2000, http://www.compaq.com/hpc/news/news_hpc_60171.html.

[6] Richard M. Fujimoto. *Parallel and Distributed Simulation Systems*, John Wiley & Sons, Inc., 2000.

[7] http://www.ssfnet.org.

[8] James H. Cowie, David M. Nicol, Andy T. Ogielski. *Modeling the Global Internet.* Computing in Science & Engineering 1(1):30-38, 1999.

[9] Jason Liu and David M. Nicol. *Dartmouth Scalable Simulation Framework User's Manual.* Dartmouth College Dept. of Computer Science, April 24, 2001.

[10] P. Dickens, M. Haines, P. Mehrotra, and D. Nicol. Towards a thread-based parallel direct execution simulator. In Proceedings of the 29th Hawaii International Conference on System Sciences (HICSS29) , pages 424–433, Jan. 1996

[11] P. Dickens (Presenter). Performance Prediction for MPI Programs Executing on Workstation Clusters. http://citeseer.nj.nec.com/291593.html.

[12] J.E. Veenstra and R.J. Fowler. *Mint: A front end for efficient simulation of shared-memory multiprocessors.* In Proc. 2nd Int. Workshop on Modeling, Analysis, and Simulation of Comp. and Telecommunication Syst. (MASCOTS) (Jan. 1994), pp. 201-207.

[13] F.J. Alexander and G.L. Eyink. *A Rayleigh-Ritz Calculation of the Effective Potential far from Equilibrium.* Physical Review Letters 78 1, 1997.